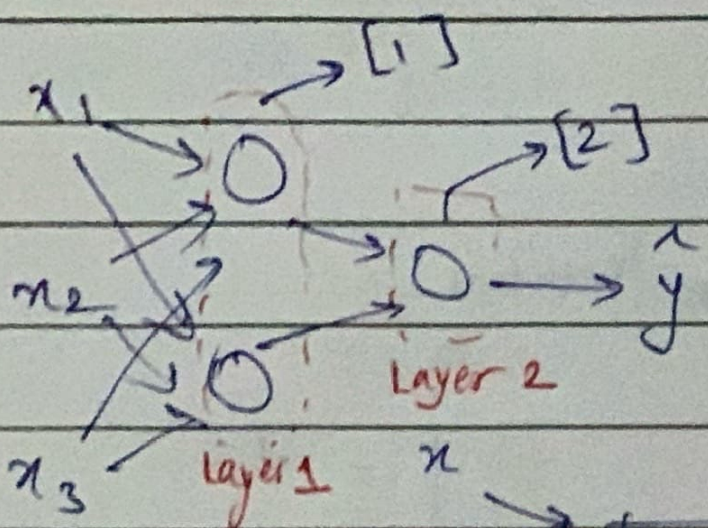
WEEK 3: Shallow Neural Networks

Date: 28/07/24



x

$w^{[1]} \rightarrow$ $\boxed{z^{[1]} = w^{[1]} x + b^{[1]}}$

$b^{[1]} \rightarrow$

Further

$\boxed{a^{[1]} = \sigma(z^{[1]})} \rightarrow \boxed{z^{[2]} = w^{[2]} a^{[2]} + b^{[2]}}$

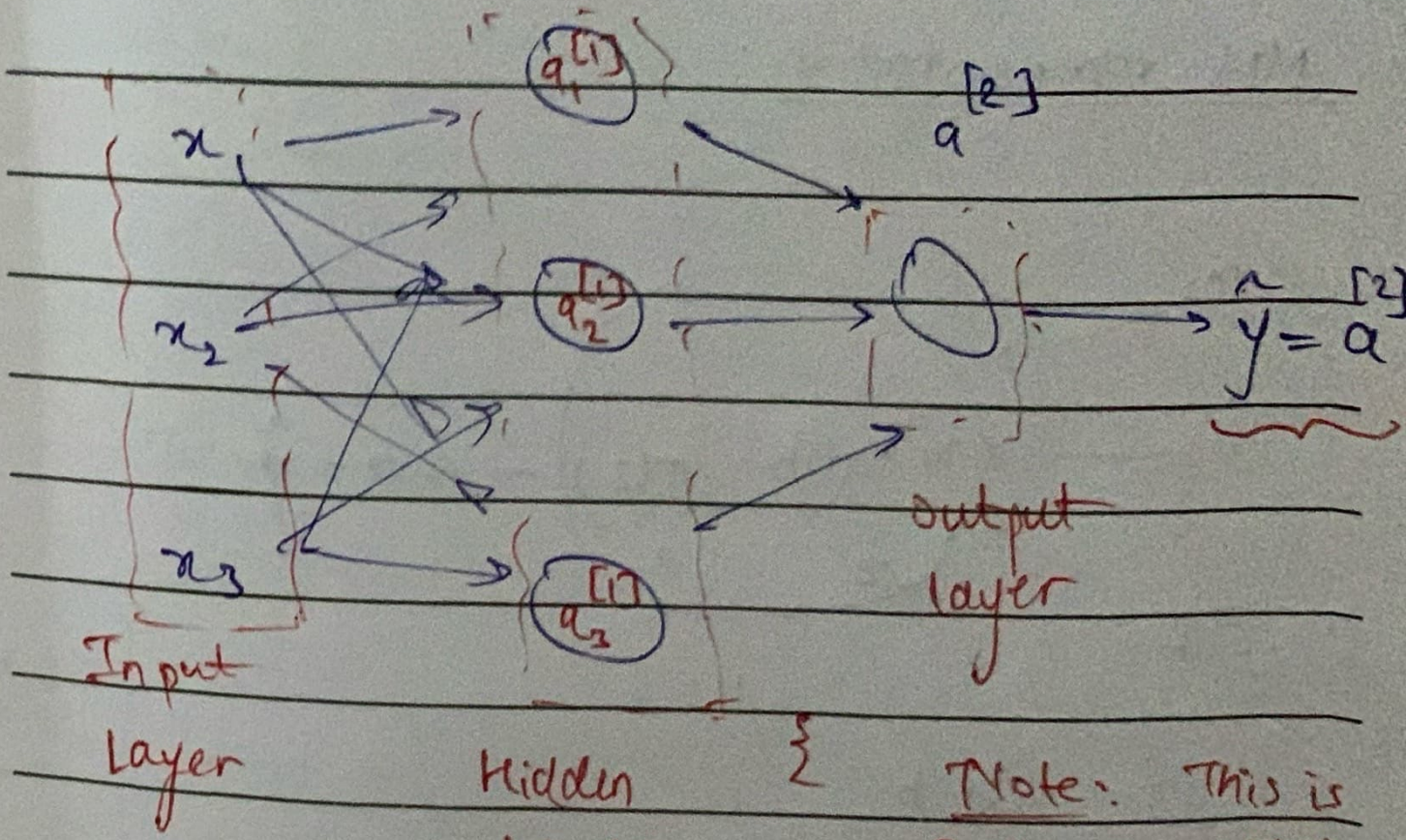
$w^{[2]} \rightarrow$

$b^{[2]} \rightarrow$

$\boxed{a^{[2]} = \sigma(z^{[2]})}$

$\boxed{L(a^{[2]}, y)}$

$a^{[0]} = X$ $a^{[1]}$ Date:

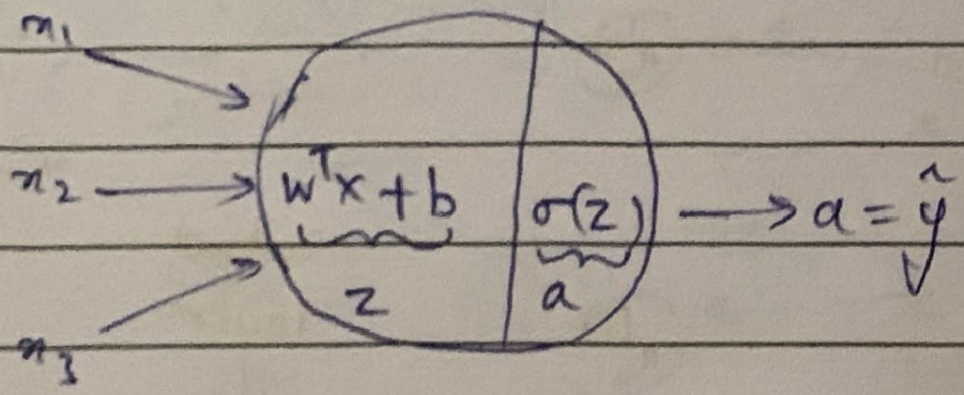


Note: This is a 2 layer NN. we don't count the input layer. }

In the hidden layer, you can't see what's there in the training set, while you can see it in the input and output layer.

Date:

NN representation:

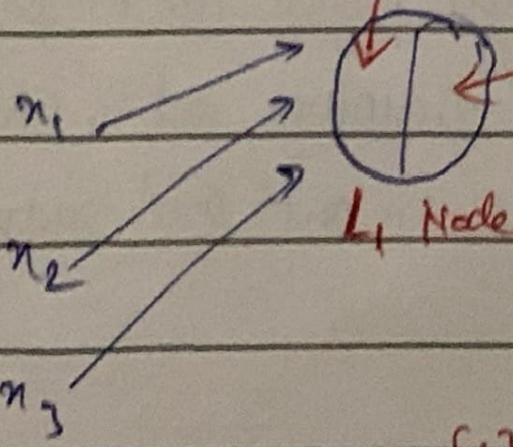


$z = w^T x + b$

$a = \sigma(z)$

$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$

$a_1^{[1]} = \sigma(z_1^{[1]})$

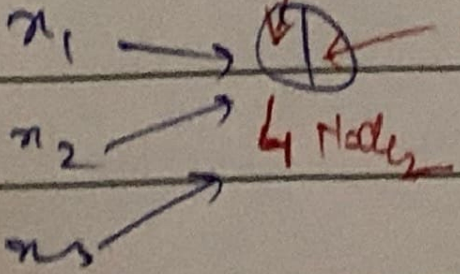


layer $[l]$

NOTE: $a_i^{[l]}$ → Node in layer

$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$

$a_2^{[1]} = \sigma(z_2^{[1]})$



Date:

Similarly,

~~four~~ for all 4 nodes in layer 1, the $z^{[1]}$ is computed.

To vectorise; $W^{[1]}$ $b^{[1]}$

$$Z^{[1]} = \begin{bmatrix} \text{---} & w_1^{[1]T} & \text{---} \\ \text{---} & w_2^{[1]T} & \text{---} \\ \text{---} & w_3^{[1]T} & \text{---} \\ \text{---} & w_4^{[1]T} & \text{---} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

(4×3)

$$= \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix}$$

$$= \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

Date:

$$z^{[1]} = w^{[1]T} x + b^{[1]}$$

$(4,1) \quad (4,3) \cdot (3,1) \quad (4,1)$

$$a^{[1]} = \sigma(z^{[1]})$$

$(4,1) \quad (4,1)$

$$z^{[2]} = w^{[2]} a^{[1]} + b^{[2]} \qquad \{w^T = w^{[2]}\}$$

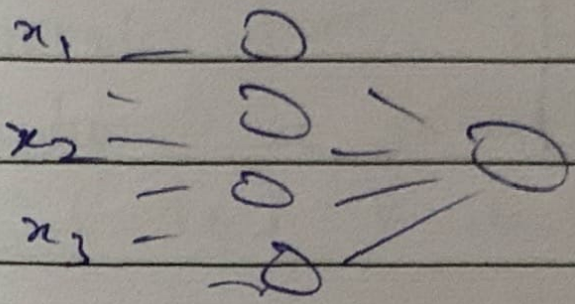
$(1,1) \quad (1,4) \quad (4,1) \quad (1,1)$

$$a^{[2]} = \sigma(z^{[2]})$$

$(1,1) \quad (1,1)$

Date:

Vectorising across multiple training examples:



$$x \longrightarrow a^{[2]} = \hat{y}$$

$$x^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)}$$

$$x^{(2)} \longrightarrow a^{[2](2)} = \hat{y}^{(2)}$$

$$\vdots \qquad \vdots$$

$$x^{(m)} \longrightarrow a^{[2](m)} = \hat{y}^{(m)}$$

$a^{[l](m)}$

- l → layer
- m → training examples

for $i = 1$ to m : ;{

$$z^{[1](i)} = w^{[1]} x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

$$z^{[2](i)} = w^{[2]} a^{[1](i)} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)})$$

}

Date:

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

(n_x, m)

$$z^{[1]} = w^{[1]} X + b^{[1]}$$

$$A^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = w^{[2]} A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(z^{[2]})$$

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

Date:

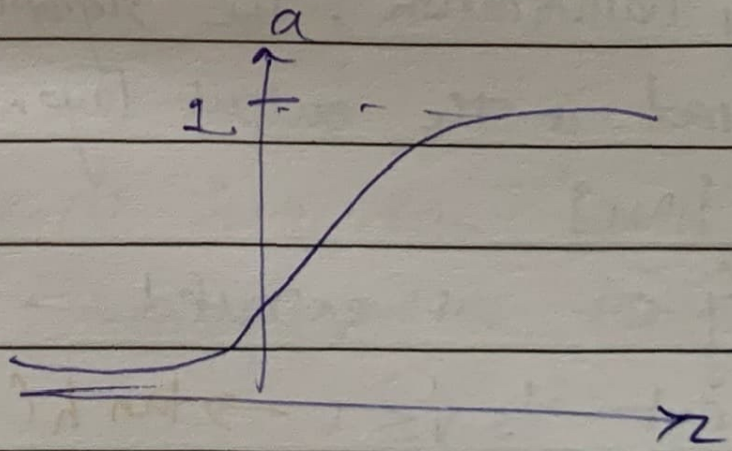
horizontally, we go across different training examples from $1 \rightarrow m$

vertically, we go across different nodes of layer $[l]$ that $1 \rightarrow$ no. of ~~nodes~~ hidden units

Activation functions:

We used sigmoid function until now, but it is not the best choice.

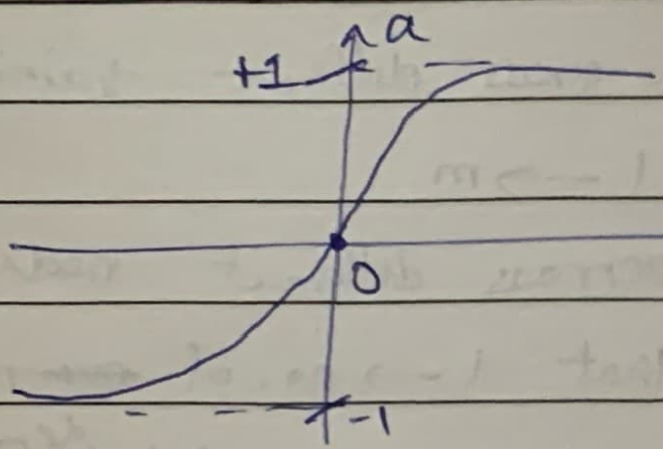
Sigmoid is a type of activation function.



$$a = \frac{1}{1 + e^{-z}} = \sigma(z)$$

Sigmoid func.

Date:



$a = \tan(z) = \frac{e^{z} + e^{-z}}{e^{z} - e^{-z}}$

tanh( ) func.

This results better by centering your data by that the mean of your data is close to zero rather than maybe 0.5, makes learning for next layer a bit easier. tan(h(x)) always better than $\sigma(x)$.

Still during binary classification, the sigmoid func. must be used in the output layer

∴ wher $\rightarrow y \{0,1\}$

Sigmoid $\leftarrow 0 \leq \hat{y} \leq 1$ is expected

and not $-1 \leq \hat{y} \leq 1 \rightarrow$ tan h( )

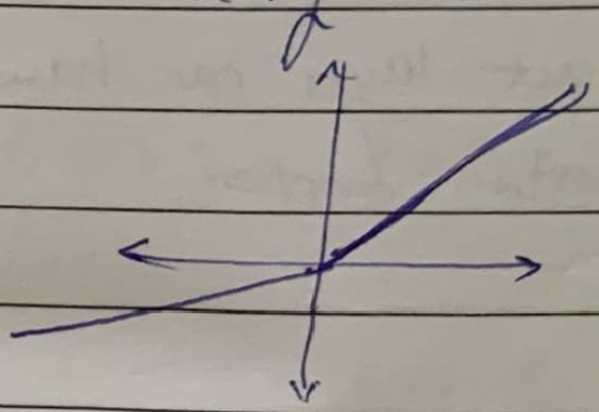
Similar problem amongst both is that is 'z' is very large (or) very small then the slope is almost zero and this slows down gradient descent.

Date:

If binary classification (0/1) → sigmoid activation function for output layer

If for hidden layers,
nothing is defined, it is most often
ReLU being used (or sometimes tanh())
problem with ReLU is that for negative
values of z, the slope is zero.
∴ Leaky ReLU can be used.

$a = \max(0, z)$



Leaky ReLU = $a = \max(0.01z, z)$

ReLU faster than 'tanh and sigmoid'
∴ learning will be faster.

ReLU / Leaky ReLU > tanh() > sigmoid

Date:

Activation function :

These are to be non-linear functions, other wise, there is no-point of the hidden layer.

If $y \in \mathbb{R}$

then hidden layers could take up ReLU, tanh so that output layer can have a linear activation function.

~~Derivations~~

Derivatives :

1) $\frac{d}{dz} g(z)$. g = sigmoid funct.

$= \frac{d}{dz}\left(\frac{1}{1+e^{-z}}\right) = \frac{+1}{(1+e^{-z})^2}\left(\right.$ $(1+e^{-z}) + e^{-2}$ $\left.\right)$

Date:

$$= \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}-1}{(1+e^{-z})^2}$$

$$= \frac{1}{1+e^{-z}} - \frac{1}{(1+e^{-z})^2}$$

$$= \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$\Rightarrow \frac{d}{dz} g(z) = (g(z))(1-g(z))$$

if $z = 10$, $g(z) \approx 1$

$$\frac{d}{dz} g(z) \approx 1(1-1) \approx 0$$

if $z = -10$, $g(z) \approx 0$

$$\frac{d}{dz} g(z) \approx 0 \times (1-0) \approx 0$$

Date:

if $z = 0$ $\quad g(z) = 1/2$

$\frac{d}{dz} g(z) = \frac{1}{2}(1 - 1/2) = 1/4$

$g'(z) = \frac{d}{dz} g(z)$

if $a = g(z)$

$\therefore g'(z) = a(1-a)$

(2) tanh( ) activation function

$\frac{d}{dz} g(z)$ = slope of $g(z)$ at z

$g(z) = \tanh(z) = \frac{e^z + e^{-z}}{e^z - e^{-z}}$

$g'(z) = 1 - (\tanh(z))^2$

Date:

$z = 10$ tanh ≈ 1

$g'(z) \approx 0$

$z = -10$ tanh ≈ -1

$g'(z) \approx 0$

$z = 0$ tanh $z = 0$

$g'(z) = 1$

$a = g(z)$, $g'(z) = 1 - a^2$

(3) ReLU and Leaky ReLU

$g(z) = \max(0, z)$

$$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

Date:

$$g(z) = \max(0.01z, z)$$

$$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

Gradient descent for NN: $[n_x = n^{[0]}, n^{[1]}, n^{[2]} = 1]$

Parameters: $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}$

$(n^{[1]}, n^{[0]})$ $(n^{[1]}, 1)$ $(n^{[2]}, n^{[1]})$ $(n^{[2]}, 1)$

Cost function:

$$J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}, y)$$

$a^{[2]}$

Date:

Gradient descent:

Repeat {

Compute predict ($\hat{y}^{(i)}$, i=1, i=2 ... m)

$dw^{[1]} = \frac{\partial J}{\partial w^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$, ...

$w^{[1]} = w^{[1]} - \alpha\, dw^{[1]}$

$b^{[1]} = b^{[1]} - \alpha\, db^{[1]}$

}

}

Date:

Formulas for computing derivatives:

→ Forward propagation:

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[1]}$$

$$A^{[2]} = g^{[2]}(Z^{[2]})$$

→ Back propagation:

$$Y = [y^{(1)}\ y^{(2)} \dots y^{(m)}]$$

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

$(n^{[1]}, m)$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis=1, keepdims=True)$$

Date:

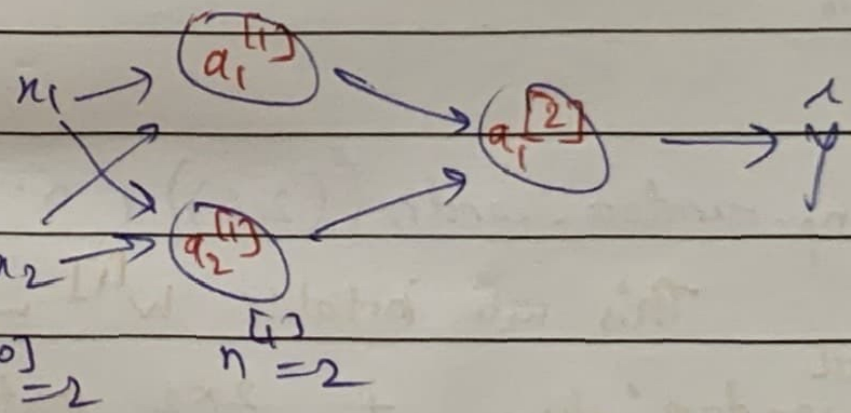
$$dz^{[1]} = \underbrace{w^{[2]T} dz^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]'}(z^{[1]})}_{(n^{[1]}, m)}$$

element-wise product

$$dw^{[1]} = \frac{1}{m} dz^{[1]} x^T$$

$$db^{[1]} = \frac{1}{m} np.sum(dz^{[1]}, axis=1, keepdims=True)$$

Random initialisation:



$$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Date:

$a_1^{[1]} = a_2^{[1]}$; $dz_1^{[1]} = dz_2^{[1]}$

$W^{[2]} = [0 \quad 0]$ $\quad dw = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$

every is same

∴ ~~Symmetric~~ ∴

The NN having hidden units all will compute the same function

∴ No point.

→ Random:

$W^{[1]} = np.random.randn((2,2)) * 0.01$

This will initialise $W^{[1]}$ with very small random values of 2×2 matrix

$b^{[1]} = np.zeros((2,1))$

$W^{[2]}$ → randomly

$b^{[2]}$ → to zeros.

Date:

The constant used is '0.01' very small
∵ if activation function used is:
tanh() (or) sigmoid ; then the
large values will give almost zero slope
and gradient descent will be very slow so
the model will learn very slowly.